

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО**

**ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«Донской государственный технический университет»**

**(ДГТУ)**

Кафедра «Программное обеспечение вычислительной техники и

автоматизированных систем»

**ЛАБОРАТОРНАЯ РАБОТА № 5**

по дисциплине «Эвристические методы и алгоритмы»

тема: «Модифицированные алгоритмы “Критического пути”»

Выполнил:

ст. гр. ВПР 31 Д. С. Кононов

Проверил:

д.н., профессор В. Г. Кобак

Ростов-на-Дону

2020

1. **Введение**

Предметом области исследования расписаний является круг задач проектирования и организационного управления в различных системах, в которых требуется найти наилучшее (оптимальное) значение выбранных критериев их функционирования с учетом имеющихся ограничений.

В данном руководстве будут рассмотрены модификации алгоритма “Критического пути” для решения задачи получения расписания в однородной вычислительной системе.

В качестве основного критерия рассматривается минимаксный критерий, который минимизирует максимальное время окончания работы программ.

# Постановка задачи

Имеется вычислительная система, состоящая из  несвязанных однородных устройств . На обработку поступает - множество независимых параллельных заданий (работ) , известно время решения **** каждого задания  на устройстве ****(****для всех устройств одинаково, если не равно бесконечности). При этом задание может выполняться на одном устройстве, время выполнения определяется значением ****, и не выполняться на другом совсем, тогда время выполнения задачи на этом устройстве определено как **= ∞**. В каждый момент времени выполняется не более одного задания на устройстве. Требуется найти такое распределение заданий по процессорам, при котором суммарное время выполнения заданий на каждом из процессоров было бы минимальным.

Минимаксный критерий определяется в следующем виде: , где - время завершения работы процессора .

**Задача №1**

Для примера рассмотрим задачу , где , . Исходная матрица заданий



# Модифицированные алгоритмы “Критического пути”

Для решения задачи планирования с матрицами загрузки имеющими бесконечности рассматривается три различные модификации алгоритма.

# Распределение без изменений

Ш.1 Упорядочиваем строки в порядке убывания веса множества заданий; , где , , элементы равные бесконечности игнорируем.

Ш.2 Начальная загрузка всех устройств равна нулю, выбирается первое устройство и первое задание из множества заданий , если ** ∞**.

В том случае, когда время выполнения назначаемой работы на текущее устройство = ∞, то берется следующее устройство, для которого ****приемлемо, иначе *i +* 1.

Ш.3 Выбираем устройство с наименьшим среди всех процессоров (т.е. процессоров с минимальным временем загрузки на текущий момент).

Ш.4 Назначить следующим заданием на процессор , пересчитать .

Ш.5 Алгоритм заканчивается с расписанием, общая длина которого равняется 

Для примера решим задачу №1

Выполнив Ш.1 матрица заданий примет вид.



Согласно Ш.2 На первое устройство назначается для выполнения задание . Расписание на 1-вом шаге:



Считаем загрузку (время завершения выполнения работы) каждого устройства ,,,. Выбираем следующее устройство с наименьшей загрузкой, т.е. , но вторая работа на втором устройстве выполняться не может ****, тогда выбираем , и ****. Расписание на 2-ом шаге примет вид:



Пересчитав загрузку каждого процессора , , , переходим к Ш.4. Назначаем следующую работу на процессор с наименьшей загрузкой :



При  назначаем следующую работу на процессор (с наименьшей загрузкой), т.к  :



Выполнив все итерации, получаем расписание



Критерий оценки алгоритма – максимальное время завершения выполнения работ:=26

# С учетом бесконечности

Ш.1 Упорядочиваем строки в порядке убывания веса множества заданий; , при этом считаем количество бесконечностей в строке и учитываем при сортировке, первыми будут строки имеющие хотя бы один элемент - бесконечность.

Последующие шаги алгоритма согласно пункта 3.1.

Алгоритм рассмотрим на примере, решив задачу №1.

Выполнив Ш.1 пункта 3.2 множество заданий примет вид



Итерационно выполнив шаги Ш.2-Ш.5, аналогично примеру, рассмотренному в пункте 3.1, получим расписание:



Результат: =21

# С учетом количества бесконечностей в строке

Ш.1 Упорядочиваем строки в порядке убывания веса множества заданий;  и в порядке убывания количества бесконечностей в строке, если строки имеют одинаковое число бесконечностей, то выполняем упорядочивание в порядке убывания элементов отличных от бесконечностей.

Последующие шаги алгоритма согласно пункта 3.1.

Решим задачу №1.

Выполнив Ш.1 пункта 3.3 множество заданий примет вид



Итерационно выполнив шаги Ш.2-Ш.5, аналогично примеру, рассмотренному в пункте 3.1, получим расписание:



Результат: =23

1. **Реализации алгоритма половинного деления множества заданий Python**

from random import randint

class Scheduling:

def \_\_init\_\_(self, n=4, left=1, right=25):

m = randint(8, 15)

print('N = 4; M =', m)

self.p = [randint(left, right) for i in range(m)]

self.mass = [[] for i in range(m)]

for i in range(len(self.p)):

inf = 4

for j in range(n):

if randint(0, 2):

self.mass[i].append(self.p[i])

inf -= 1

else:

self.mass[i].append(0)

if inf == 4:

self.mass[i][randint(0, 3)] = self.p[i]

def sort(self, mass=None):

if not mass:

mass = self.mass

mass = mass.copy()

sorted\_mass = list(sorted(self.p))

new\_mass = []

for i in sorted\_mass:

for j in mass:

if i in j:

new\_mass.append(j)

mass.pop(mass.index(j))

return list(reversed(new\_mass))

def take\_result(self, mass):

print('Tmax = max(f(P1), f(P2), f(P3)',

' f(P4)) = max({}) = {}'.format(

mass[[sum(i) for i in mass].

index(max([sum(i) for i in mass]))],

max([sum(i) for i in mass])))

print('Tmax = P{} = {}'.format([sum(i) for i in mass].

index(max([sum(i) for i in mass])),

max([sum(i) for i in mass])))

def modif\_step\_1\_to\_3(self, mass):

tmp = []

for i in mass:

k = 0

for j in i:

if j == 0:

k += 1

tmp.append([k, i])

new\_mass = []

for i in reversed(range(len(mass[0]))):

for j in tmp:

if i == j[0]:

new\_mass.append(j[1])

return new\_mass

def modif\_step\_1\_to\_2(self, mass):

tmp = []

for i in mass:

k = 0

for j in i:

if j == 0:

k += 1

tmp.append([k, i])

new\_mass = []

for j in tmp:

if j[0] > 0:

new\_mass.append(j[1])

for j in tmp:

if j[0] == 0:

new\_mass.append(j[1])

return new\_mass

def infinity(self):

self.output\_mass(self.sort())

mass = self.modif\_step\_1\_to\_3(self.sort())

self.output\_mass(mass)

self.main(mass)

def low\_infinity(self):

self.output\_mass(self.sort())

mass = self.modif\_step\_1\_to\_2(self.sort())

self.output\_mass(mass)

self.main(mass)

def critical(self):

print('Descending sorted:')

mass = self.sort(self.mass)

self.output\_mass(mass)

self.main(mass)

def main(self, mass):

p = [[] for i in range(len(self.mass[0]))]

for i in mass:

tmp = [sum(i) for i in p]

for j in range(len(i)):

if i[j] != 0:

tmp[j] += i[j]

else:

tmp[j] = None

m = 99999

for k in tmp:

if k:

if k < m:

m = k

m = tmp.index(m)

p[m].append(tmp[m] - sum(p[m]))

for i in range(len(p)):

print('P{} ='.format(i), p[i], '| sum =', sum(p[i]))

self.take\_result(p)

def output\_mass(self, mass):

string = ''

for i in mass:

for j in i:

if len(str(j)) == 1:

if j == 0:

string = '{} oo '.format(string)

else:

string = '{} {} '.format(string, j)

else:

string = '{} {} '.format(string, j)

string = '{}\n'.format(string)

print(string)

def \_\_str\_\_(self):

string = ''

for i in self.mass:

for j in i:

if len(str(j)) == 1:

string = '{} {} '.format(string, j)

else:

string = '{} {} '.format(string, j)

string = '{}\n'.format(string)

return string

if \_\_name\_\_ == '\_\_main\_\_':

a = Scheduling()

print(a)

a.critical()

a.low\_infinity()

a.infinity()

1. **Результаты, выводы**

В ходе лабораторной работы была написана программа, реализующая модифицированные алгоритмы критического пути множества заданий, такие как: распределение без изменений, с учётом бесконечности, с учётом бесконечности в строке. В которой количество процессоров и количество заданий было определено из вариантов заданий для вариантов из пункта 4 методчики по теме «Теория расписаний». Матрица генерируется рандомно.

Также было произведено сравнение работы алгоритмов при сортированном списке заданий. Выяснили, что алгоритм позволяет получить точной результат, при использовании варианта распределение без изменений. Остальные результаты являются приближенными.

Результат работы программы:





